Estruturas de Dados - ESP412

Profª Ana Carolina Sokolonski

Bacharelado de Sistemas de Informação
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

November 19, 2024

Estruturas de Dados

1 Vetores
- Recursividade
- Remoção
- Inserção
- Busca e Remoção

2 Referências

Recursividade

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Dizemos que esses problemas têm estrutura recursiva. Para resolver um problema desse tipo, podemos aplicar o seguinte método:

Recursividade

Recursão e Algoritmos Recursivos

Muitos problemas computacionais têm a seguinte propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

Dizemos que esses problemas têm estrutura recursiva. Para resolver um problema desse tipo, podemos aplicar o seguinte método:

- se a instância for pequena, resolva-a diretamente;
- senão, reduza-a a uma instância menor $\xrightarrow{entao}$ aplique o método à instância menor $\xrightarrow{entao}$ volte à instância original.

Recursividade

Busca Recursiva em Vetor

O programador só precisa mostrar como obter uma solução da instância original a partir de uma solução da instância menor; o computador faz o resto. A aplicação desse método produz um algoritmo recursivo.

Recursividade

Busca Recursiva em Vetor

O programador só precisa mostrar como obter uma solução da instância original a partir de uma solução da instância menor; o computador faz o resto. A aplicação desse método produz um algoritmo recursivo.

```
int busca_r (int x, int n, int v[]) {
    if (n == 0) return -1;
    if (x == v[n-1]) return n-1;
    return busca_r (x, n-1, v);
}
```

Exercícios:

1. Faça um algoritmo recursivo que encontre qual o maior valor de um vetor.
2. Faça um algoritmo recursivo que calcule o fatorial de um número e mostre suas parcelas. Ex:
   $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$
3. A função de Fibonacci é definida assim: $Fib(0) = 0$, $Fib(1) = 1$ e $Fib(n) = Fib(n-1) + Fib(n-2)$ para $n > 1$. Descreva a função Fib em linguagem C. Faça uma versão recursiva e outra iterativa.

Exercícios:

1.

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam] > maior)
        return v[tam];
    else
        return maior;
}
```

Exercícios:

1.

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam-1] > maior)
        return v[tam-1];
    else
        return maior;
}

int main (void){
int v1[]={134,3,234,7,567,5,678,2,899,0};
int v[100];
int tam, d=0;

for(tam=0;d!=-1;tam++){
    printf("Digite os elementos do vetor, digite -1 para sair\n");
    scanf("%d",&d);
    if(d!=-1)
     v[tam] = d;
}

  int i = maior_r(tam,v);
  printf("Maior item do Vetor= %d\n", i);
}
```

Exercícios:

2.

```c
#include <stdio.h>
#include <stdlib.h>

int fat (int n){
    if (n == 1){
        printf("1 = ");
        return 1;
    }
    else{
        printf("%d*",n);
        return (n * fat(n-1));
    }
}

int main (void){
int n;

  printf("Informe o número que você deseja saber o fatorial:");
  scanf("%d",&n);
  printf("%d! = ",n);
  printf("%d\n", fat(n));
}
```

Exercícios:

3.

```c
void fibonacci (int n){
    int fib, fib1, fib2, i;

    if (n == 0) {
        printf("0\n");
    } else if (n == 1){
        printf("0, 1\n");
    } else{
        printf("0, 1, ");
        fib1 = 1;
        fib2 = 0;
        i = 2;
        while (i <= n){
            fib = fib1+fib2;
            if (i==n) printf("%d\n", fib);
            else printf("%d, ", fib);
            fib2 = fib1;
            fib1 = fib;
            i++;
        }
    }

}
```

Exercícios:

3.

```c
/**A função de Fibonacci é definida assim:   F(0) = 0,   F(1) = 1   e   F(n) = F(n-1) + F(n-2)
para n > 1. Descreva a função F em linguagem C. Faça uma versão recursiva e uma iterativa.**/
int fibonacci (int n){
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibonacci(n-1) + fibonacci(n-2);
}
int main (void){
int n, fib, i;

  printf("Informe o número de fatores das sequências de Fibonacci:");
  scanf("%d",&n);
  for (i = 0; i <= n; i++)
     printf("Parcela %d da sequência = %d\n",i,fibonacci(i));
}
```

Vetor

Remoção:

Voltando à Estrutura de Dados Simples chamada Vetor, suponhamos que desejamos remover um elemento de índice k do vetor $v[0 \ldots (n-1)]$. Para isso, deslocamos os elementos $v[(k+1) \ldots (n-1)]$ do vetor para as posições $[k \ldots (n-2)]$.

Vetor

Remoção:

Voltando à Estrutura de Dados Simples chamada Vetor, suponhamos que desejamos remover um elemento de índice k do vetor $v[0 \ldots (n-1)]$. Para isso, deslocamos os elementos $v[(k+1) \ldots (n-1)]$ do vetor para as posições $[k \ldots (n-2)]$.

Por exemplo, o resultado da remoção do elemento de índice 3 do vetor $\{0,\ 11,\ 22,\ 33,\ 44,\ 55\}$ é o vetor $\{0,\ 11,\ 22,\ 44,\ 55\}$.

Vetor

Remoção:

Voltando à Estrutura de Dados Simples chamada Vetor, suponhamos que desejamos remover um elemento de índice k do vetor $v[0 \ldots (n-1)]$. Para isso, deslocamos os elementos $v[(k+1) \ldots (n-1)]$ do vetor para as posições $[k \ldots (n-2)]$.

Por exemplo, o resultado da remoção do elemento de índice 3 do vetor $\{0,\ 11,\ 22,\ 33,\ 44,\ 55\}$ é o vetor $\{0,\ 11,\ 22,\ 44,\ 55\}$.

É claro que o problema faz sentido se, e somente se, $0 \leq k < n$.

Vetor - Remoção

A seguinte função resolve o problema e devolve o valor do elemento removido:

```
int remover(int k, int n, int v[]){
    int x = v[k];
    for (int j = k+1; j < n; ++j)
        v[j-1] = v[j];
    return x;
}
```

Vetores - Remoção:

Note que o algoritmo funciona bem quando $k = (n - 1)$ e quando $k = 0$. A remoção consumirá mais tempo quando for menor o k.

Como usar a função? Para remover o elemento de índice 51 de $v[0 \ldots (n - 1)]$ (estou supondo que $(51 < n)$), por exemplo, basta dizer:

$x = remover(51, n, v)$;
$n- = 1$; //atualiza o valor de n

Vetores

Remoção - Versão Recursiva:

É um bom exercício escrever uma versão recursiva de "remover".
O tamanho de uma instância do problema é medido pela diferença $(n - k)$ e a instância é considerada pequena se $(n - k) = 1$.
Portanto, a base da recursão é o caso em que $k = (n - 1)$.
[Cormen et al. 2009]

Vetores

Remoção - Versão Recursiva:

É um bom exercício escrever uma versão recursiva de "remover". O tamanho de uma instância do problema é medido pela diferença $(n - k)$ e a instância é considerada pequena se $(n - k) = 1$. Portanto, a base da recursão é o caso em que $k = (n - 1)$. [Cormen et al. 2009]

```
int remover_r (int k, int n, int v[]) {
    int x = v[k];
    if (k < n-1) {
        int y = remover_r (k+1, n, v);
        v[k] = y;
    } return x;
}
```

Vetores - Inserção:

Dado um vetor de números $v[0 \ldots (n-1)]$, queremos inserir um novo número x entre os elementos de índices $(k-1)$ e k. Isso faz sentido não só quando $1 \leq k \leq (n-1)$ como também quando $k = 0$ (insere no início) e quando $k = n$ (insere no fim). Em suma, faz sentido para qualquer k no conjunto $[0 \ldots n]$.

Vetores - Inserção:

Dado um vetor de números $v[0 \ldots (n-1)]$, queremos inserir um novo número x entre os elementos de índices $(k-1)$ e k. Isso faz sentido não só quando $1 \leq k \leq (n-1)$ como também quando $k = 0$ (insere no início) e quando $k = n$ (insere no fim). Em suma, faz sentido para qualquer k no conjunto $[0 \ldots n]$.

É claro que você só deve inserir x se tiver certeza de que o vetor não está cheio; caso contrário, teremos um transbordamento (*overflow*). Portanto, certifique-se de que $(n+1) \leq N$ antes de chamar a função.

Vetores - Inserção:

A função inserir funciona mesmo quando a inserção é no início ou no fim! Para inserir um novo elemento com valor 999 entre as posições 50 e 51 (supondo $(51 \leq n)$) basta dizer:

$inserir(51, 999, n, v)$;
$n++$; // atualiza n

```
// Esta função insere x entre as posições k-1 e k do vetor v[0..n-1]
// supondo que 0 <= k <= n.
void inserir (int k, int x, int n, int v[]){
   for (int j = n; j > k; --j)
      v[j] = v[j-1];
   v[k] = x;
}
```

Inserção - Versão recursiva:

```
4  // Esta função insere x entre as posições k-1 e k do vetor v[0..n-1]
5  // supondo que 0 <= k <= n.
6  void inserir_r (int k, int x, int n, int v[]) {
7     if (k == n)  v[n] = x;
8     else {
9        v[n] = v[n-1];
10       inserir_r (k, x, n - 1, v);
11    }
12 }
```

Vetores - Busca e Remoção:

Considere uma combinação dos problemas de busca e remoção. Suponha que queremos remover todos os números 2 do vetor $v[0 \ldots (n-1)]$. É claro que o problema faz sentido com qualquer $(n \geq 0)$.

Vetores - Busca e Remoção:

Considere uma combinação dos problemas de busca e remoção. Suponha que queremos remover todos os números 2 do vetor $v[0 \ldots (n-1)]$. É claro que o problema faz sentido com qualquer $(n \geq 0)$.

Por exemplo, se $n = 7$ e $v[0 \ldots 6]$ é $\{11,\ 2,\ 22,\ 88,\ 2,\ 66,\ 33\}$, então o vetor resultante deve ser $\{11,\ 22,\ 88,\ 66,\ 33\}$. Embora o enunciado do problema não peça isso, explicitamente, vamos exigir que a função devolva o número de elementos do vetor depois da remoção.

Vetores - Busca e Remoção:

```c
// Esta função elimina todos os elementos 2 de v[0..n-1].
// Supõe apenas que n >= 0. A função deixa o resultado em
// v[0..i-1]  e devolve i.
int buscaRetira2 (int n, int v[]) {
   int i = 0;
   for (int j = 0; j < n; ++j)
      if (v[j] != 2)
         v[i++] = v[j];
   return i;
}
```

Vetores - Busca e Remoção:

```c
// Esta função elimina todos os elementos 2 de v[0..n-1].
// Supõe apenas que n >= 0. A função deixa o resultado em
// v[0..i-1]  e devolve i.
int buscaRetira2 (int n, int v[]) {
   int i = 0;
   for (int j = 0; j < n; ++j)
      if (v[j] != 2)
         v[i++] = v[j];
   return i;
}
```

A instrução $v[i++] = v[j]$ tem o mesmo efeito de $v[i] = v[j]; ++i;$. No início de cada iteração: $v[0 \ldots (i-1)]$ é o vetor resultante da remoção dos elementos de $v[0 \ldots (j-1)]$; óbvio que $i \leq j$.

Busca e Remoção - Recursivo:

Note como a instrução $v[m] = v[n-1]$ coloca $v[n-1]$ no seu lugar definitivo. A função "*buscaRetira2_r*" elimina todos os números 2 de $v[0 \ldots (n-1)]$. A função deixa o resultado em $v[0 \ldots (i-1)]$ e devolve i.

```
int buscaRetira2_r (int n, int v[]) {
    if (n == 0) return 0;
    int m = buscaRetira2_r (n - 1, v);
    if (v[n-1] == 2) return m;
    v[m] = v[n-1];
    return m + 1;
}
```

Referências

Referências


CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.